New trends in coupled simulations featuring domain decomposition and metacomputing

Philippe d'Anfray*
Laurence Halpern§
Juliette Ryan¶

Abstract. *In this paper we test the feasibility of coupling two heterogeneous mathematical modeling integrated within two different codes residing on distant sites. A prototype is developed using Schwarz type domain decomposition as the mathematical tool for coupling. The computing technology for coupling uses a `CORBA` environment to implement a distributed client-server programming model. Domain decomposition methods are well suited to reducing complex physical phenomena into a sequence of parallel subproblems in time and space. The whole process is easily tuned to underlying hardware requirements.*

Résumé. *Dans ce travail nous testons la possibilité de coupler deux modèles physiques distincts intégrés dans deux codes différents, implantés en deux lieux distants. Nous développons un prototype, qui repose sur une méthode de décomposition de domaines de type Schwarz stationnaire ou évolutif. La technologie informatique utilise un environnement `CORBA` pour mettre en œuvre un modèle de programmation distribuée client-serveur. Grâce à la décomposition de domaine en temps, l'algorithme a une granularité modulable, s'adaptant ainsi aux spécifités de l'environnement informatique.*

M2AN, Vol. 36, N°5, pp. 953-970 (September/October 2002)

*`CEA DTI-CISC` 91191 Gif sur Yvette Cedex France (on leave from `ONERA DTIM-CHP`) and `LAGA`, Université Paris XIII, 93430 Villetaneuse France. `anfray@cea.fr`

§LAGA, Université Paris XIII, 93430 Villetaneuse France.`halpern@math.univ-paris13.fr`

¶`ONERA DTIM-CHP` and `LAGA`, Université Paris XIII, 93430 Villetaneuse France. `ryan@math.univ-paris13.fr`

1 Introduction

Within the realm of partial differential equations, multifields models can quite significantly reduce the problem complexity while increasing the numerical accuracy. The complexity may be due to heterogeneous physics, mathematics, discretisations, or heterogeneous computing environments.

A solution to the physical and mathematical aspect is domain decomposition methods, well suited to reducing complex physical phenomena into a sequence of subproblems in time and space easier to model and which to some extent can be solved simultaneously.

Many tools have been developed recently to achieve such cooperative computing that often require thorough modifications of codes to integrate new functionalities for coupling and much thinking as to where and who is in charge of the coupling. The goal of the `CORBA` standards developed by the `OMG` (Object Management Group) is to simplify these procedures.

Recent extensions of Schwarz type domain decomposition to time decomposition allow greater flexibility in terms of granularity of the computation. Time-space domains can be adjusted to the underlying computing hardware and networks.

The aim of this paper is to test the feasibility of this process, applied to heterogeneous mathematical modeling of the convection diffusion problem, running on different computing sites with the use of, and monitored by, the `CORBA` environment.

The first part of this paper describes the time and space domain decomposition technique involved, the second part is a description of the `CORBA` technology. Finally an application is presented with the case of a convection diffusion problem around an airfoil. The domain is split into 2 subdomains, finite elements round the airfoil and finite differences in the far field, each domain being computed with two different codes coupled via `CORBA`.

2 Tools in domain decomposition methods

Let $(0,T)$ be a bounded time interval, and Ω a domain in $\mathbb{R}^2$ with boundary $\Gamma = \partial\Omega$. Let us consider the convection diffusion equation with variable coefficients,

$$\mathcal{P}U \equiv \frac{\partial U}{\partial t} + \mathbf{a} \cdot \nabla U - \nu \Delta U = 0 \text{ in } \Omega \times (0,T), \tag{1}$$

associated with the initial data $U(\cdot,0) = U^0$ in Ω at time 0 and the Dirichlet boundary condition $U = 0$ on $\Gamma = \partial\Omega$. The viscosity ν is a strictly positive coefficient. The advection velocity $\boldsymbol{a}$ is a smooth function of x, in our case always non-zero in Ω.

A general technique to solve the latter is to first discretise in time with an implicit scheme such as the second order Crank-Nicolson scheme :

$$\frac{u^{n+1} - u^n}{\Delta t} + \frac{1}{2}[\mathbf{a} \cdot \nabla u^{n+1} - \nu \Delta u^{n+1}] + \frac{1}{2}[\mathbf{a} \cdot \nabla u^n - \nu \Delta u^n] = 0 \text{ in } \Omega, \tag{2}$$

where u^n is an approximation of $U(\cdot, t_n)$. At each step n , u^{n+1} is solution of a steady equation such as $\mathcal{L}u = F$ where

$$\mathcal{L} \equiv \frac{2}{\Delta t} + \mathbf{a} \cdot \nabla - \nu \Delta$$

and $F = (\frac{2}{\Delta t} - \mathbf{a} \cdot \nabla + \nu \Delta) u^n$.

In the rest of this paper, the quantity $\frac{2}{\Delta t}$ will be denoted by c.

Fast solvers for steady problems can now be applied such as non-overlapping domain decomposition methods using Schwarz-type algorithms designed in [5] which are presented in section 2.1. Algorithms global in time, namely Schwarz waveform relaxation algorithms developed in [2] will be presented in section 2.2.

2.1 Steady problems

As stated previously, we are led to the problem

$$\begin{cases} \mathcal{L}u \equiv (c + \mathbf{a} \cdot \nabla - \nu \Delta) u = F \text{ in } \Omega \\ u = 0 \text{ on } \Gamma = \partial \Omega \end{cases} \tag{3}$$

Let us split Ω in two subdomains : $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$ and denote $\Gamma_{12} = \bar{\Omega}_1 \cap \bar{\Omega}_2$. On Γ_{12} , there are two unit normals at each point: $\mathbf{n_1}$ is the outgoing normal to Ω_1 , $\mathbf{n_2}$ the outgoing normal to Ω_2 and $\mathbf{n_1} + \mathbf{n_2} = \mathbf{0}$. Γ_1 will denote the part of Γ bounding Ω_1 and Γ_2 that of Γ bounding Ω_2. We shall always suppose in the theoretical part that the flow is constant and entering Ω_2, *i.e.* that $\mathbf{a} \cdot \mathbf{n_1} > \mathbf{0}$, although the strategy works for more general situations (see [5]).

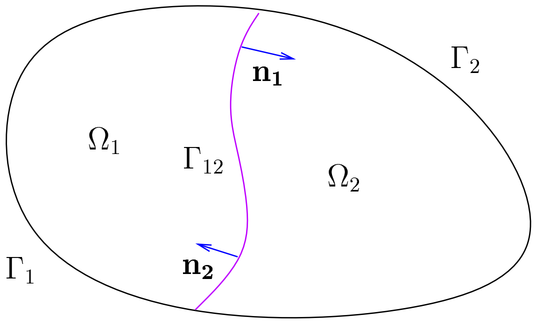


Figure 1: non-overlapping 2 domains decomposition

The problem is equivalent to the following coupling problem where u_1 is the restriction of u to Ω_1 and u_2 the restriction of u to Ω_2.

$$\begin{cases} \mathcal{L}u_1 = F \text{ in } \Omega_1 \\ \mathcal{L}u_2 = F \text{ in } \Omega_2 \\ u_1 = 0 \text{ on } \Gamma_1 \\ u_2 = 0 \text{ on } \Gamma_2 \end{cases} \qquad \begin{array}{l} u_1 = u_2 \text{ on } \Gamma_{12} \\ \dfrac{\partial u_1}{\partial n_1} + \dfrac{\partial u_2}{\partial n_2} = 0 \text{ on } \Gamma_{12} \end{array}$$

Let us now identify the transparent boundary condition for Ω_1. For any g defined on Γ_{12}, we introduce the problem

$$\begin{cases} \mathcal{L}w = 0 \text{ in } \Omega_2 \\ w = 0 \text{ on } \Gamma_2 \\ w = g \text{ on } \Gamma_{12} \end{cases}$$

The Dirichlet-Neumann operator for Ω_2, $\mathcal{K}_1^T$ is defined as

$$\mathcal{K}_1^T g = \frac{\partial w}{\partial n_2}$$

Thus any solution w of the boundary problem in Ω_2

$$\begin{cases} \mathcal{L}w = 0 \text{ in } \Omega_2 \\ w = 0 \text{ on } \Gamma_2 \end{cases}$$

is also solution on Γ_{12} of :

$$\frac{\partial w}{\partial n_2} - \mathcal{K}_1^T w = 0$$

The operator $\mathcal{K}_2^T$ is similarly defined : we introduce the problem in Ω_1

$$\begin{cases} \mathcal{L}v = 0 \text{ in } \Omega_1 \\ v = 0 \text{ on } \Gamma_1 \\ v = g \text{ on } \Gamma_{12} \end{cases}$$

The Dirichlet-Neumann operator for Ω_1, $\mathcal{K}_2$ is given by

$$\mathcal{K}_2^T g = \frac{\partial v}{\partial n_1}$$

Any solution v of the boundary problem in Ω_1

$$\begin{cases} \mathcal{L}v = 0 \text{ in } \Omega_1 \\ v = 0 \text{ on } \Gamma_1 \end{cases}$$

is also solution on Γ_{12} of

$$\frac{\partial v}{\partial n_1} - \mathcal{K}_2^T v = 0$$

We define the operators

$$\mathcal{B}_1^T = \frac{\partial}{\partial n_1} + \mathcal{K}_1^T$$
$$\mathcal{B}_2^T = \frac{\partial}{\partial n_2} + \mathcal{K}_2^T$$

For each j, $\mathcal{B}_j^T$ is the **exact transmission operator**, also named **transparent operator** for Ω_j, since by imposing it on Γ_{12}, the problem in Ω is reduced to Ω_j.

Let us now introduce the non-overlapping Schwarz algorithm.

$$\begin{cases} \mathcal{L}v^{k+1} = F, \quad \text{in } \Omega_1 \\ v^{k+1} = 0 \text{ on } \Gamma_1 \\ \mathcal{B}_1 v^{k+1} = \mathcal{B}_1 w^k \text{ on } \Gamma_{12} \end{cases} \quad \begin{cases} \mathcal{L}w^{k+1} = F, \quad \text{in } \Omega_2 \\ w^{k+1} = 0 \text{ on } \Gamma_2 \\ \mathcal{B}_2 w^{k+1} = \mathcal{B}_2 v^k \text{ on } \Gamma_{12} \end{cases} \tag{4}$$

where $\mathcal{B}_1$ et $\mathcal{B}_2$ are any boundary operators of the form $\mathcal{B}_1 = \partial_{n_1} + \mathcal{K}_1$, $\mathcal{B}_2 = \partial_{n_2} + \mathcal{K}_2$, and $\mathcal{K}_1$ and $\mathcal{K}_2$ are differential operators in the tangential variable. .

The error $(V^k = v^k - u, W^k = w^k - u)$ is solution of an homogeneous problem (*i.e.* F=0) and is thus identically null at step 2 if the boundary operators satisfy

$$\mathcal{B}_2 V^1 = \mathcal{B}_1 W^1 = 0 \text{ on } \Gamma_{12}$$

which is achieved when $\mathcal{B}_1$ and $\mathcal{B}_2$ are the transparent operators, *i.e.*if $\mathcal{B}_1 \equiv \mathcal{B}_1^T$ and $\mathcal{B}_2 \equiv \mathcal{B}_2^T$.

This remark is the basis of the analysis in [7]. These are integral operators, and differential operators would be preferable. Therefore they are approximated (as in the absorbing boundaries techniques) through a generalised local study.

Let $\Omega = \mathbb{R}^2$, and Ω_1 and Ω_2 be two half spaces delimited by $\Gamma_{12} = \{(x, y), \alpha x + \beta y = 0\}$. For simplicity we shall write $\mathbf{n} = \mathbf{n_1} = (\alpha, \beta)$, so that $\mathbf{n_2} = -\mathbf{n}$. We shall perform the change of variables $(X = \alpha x + \beta y, Y = -\beta x + \alpha y)$, and the two half spaces become $\Omega_1 = \{(X, Y), X < 0\}$ and $\Omega_2 = \{(X, Y), X > 0\}$ as in Figure 2.

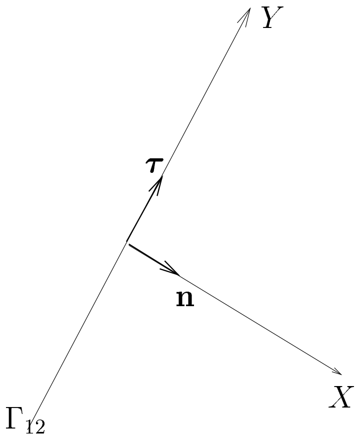


Figure 2: non-overlapping half spaces

$\boldsymbol{\tau}$ denotes the unit vector tangent to the interface such that $(\boldsymbol{n}, \boldsymbol{\tau}) = +\pi/2$.

Operator K_2^T is defined in the new variables by

$$\mathcal{K}_2^T = \frac{\partial V}{\partial X}$$

where V is given by

$$\begin{cases} \mathcal{L}V = (c + \boldsymbol{a} \cdot \boldsymbol{n}\, \partial_X + \boldsymbol{a} \cdot \boldsymbol{\tau}\, \partial_Y - \nu \Delta)V = 0, \quad X < 0, \\ V = g \text{ at } X = 0, \end{cases}$$

A partial Fourier transformation along the Y variable is

$$\hat{u}(X, \xi) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} u(X, Y) e^{-i\xi Y} dY$$

The Fourier transform of V is solution of the ordinary differential equation (ODE) in the X variable, where ξ appears as a parameter

$$\begin{cases} -\nu \dfrac{\partial^2 \hat{V}}{\partial X^2} + \boldsymbol{a} \cdot \boldsymbol{n} \dfrac{\partial \hat{V}}{\partial X} + (c + i\boldsymbol{a} \cdot \boldsymbol{\tau}\xi + \nu \xi^2)\hat{V} = 0, X < 0, \\ \hat{V} = \hat{g} \text{ at } X = 0, \end{cases}$$

The characteristic polynomial associated to this ODE is :

$$P(i\xi;\lambda) = -\nu\lambda^2 + \boldsymbol{a}\cdot\boldsymbol{n}\lambda + (c + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2)$$

Its discriminant is given by:

$$\boldsymbol{a}\cdot\boldsymbol{n}^2 + 4\nu(c + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2) \tag{5}$$

The real part of (5) is strictly positive. Let ρ be the square root of (5) with a strictly positive real part :

$$\rho^2(i\xi) = a^2 + 4\nu(c + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2), \quad \Re\rho > 0.$$

Furthermore we have $\Re\rho > \boldsymbol{a}\cdot\boldsymbol{n}$. The polynomial $P(i\xi;\cdot)$ has two complex roots :

$$\lambda_\pm(i\xi) = \frac{\boldsymbol{a}\cdot\boldsymbol{n} \pm \rho}{2\nu}$$

It is straightforward to see that $\Re\lambda_+(i\xi) > 0$ and $\Re\lambda_-(i\xi) < 0$. Thus, using similar arguments as in [3], there exists a function $\alpha(\xi)$, locally integrable, such that

$$\hat{V} = \alpha e^{\lambda_+ X} \tag{6}$$

Therefore V satisfies

$$\frac{\partial\hat{V}}{\partial X} = \lambda_+\hat{V} \text{ for } X < 0.$$

The operator $\mathcal{K}_2^T$ on the boundary is given by

$$\mathcal{K}_2^T(\frac{\partial}{\partial Y}) = \lambda_+(\frac{\partial}{\partial Y}) \tag{7}$$

In the same manner the operator $\mathcal{K}_1^T$ is given by

$$\mathcal{K}_1^T(\frac{\partial}{\partial Y}) = -\lambda_-(\frac{\partial}{\partial Y})$$

Hence operators $\mathcal{B}_1^T$ et $\mathcal{B}_2^T$ are defined by

$$\begin{aligned} \mathcal{B}_1^T &= \frac{\partial}{\partial X} + \mathcal{K}_1^T(\frac{\partial}{\partial Y}) = \frac{\partial}{\partial n} - \lambda_-(\frac{\partial}{\partial \tau}) \\ \mathcal{B}_2^T &= -\frac{\partial}{\partial X} + \mathcal{K}_2^T(\frac{\partial}{\partial Y}) = -\frac{\partial}{\partial n} + \lambda_+(\frac{\partial}{\partial \tau}) \end{aligned}$$

In the absorbing boundaries theory developed in [3], rational approximations of $\lambda_\pm$ were made for small values of ν. Nataf in [7] concerning adapted transmission conditions, performed a polynomial approximation for small values of ξ. In this paper , we present the polynomial approximation developed in [4] optimizing the reflection coefficient over a range of frequencies.

Let us look back at the iterative algorithm (4) in the two half-spaces, with operators

$$\begin{aligned} \mathcal{B}_1 &= \frac{\partial}{\partial n} + \mathcal{K}_1(\frac{\partial}{\partial \tau}) \\ \mathcal{B}_2 &= -\frac{\partial}{\partial n} + \mathcal{K}_2(\frac{\partial}{\partial \tau}) \end{aligned}$$

The error is solution of

$$\begin{cases} \mathcal{L}V^{k+1} = 0, \quad \text{for } X < 0, \\ (\partial_X + \mathcal{K}_1(\partial_Y))V^{k+1} = (\partial_X + \mathcal{K}_1(\partial_Y))W^k, \text{ for } X = 0 \end{cases}$$

$$\begin{cases} \mathcal{L}W^{k+1} = 0, \quad \text{for } X > 0 \\ (-\partial_X + \mathcal{K}_2(\partial_Y))W^{k+1} = (-\partial_X + \mathcal{K}_2(\partial_Y))V^k, \text{ for } X = 0 \end{cases}$$

With the explicit solutions as in (6), the solution is immediately obtained through a Fourier transform

$$\begin{cases} \hat{V}^k = \alpha^k e^{\lambda_+ x}, \hat{W}^k = \beta^k e^{\lambda_- x} \\ (\lambda_+ + \mathcal{K}_1(i\xi))\alpha^{k+1} = (\lambda_- + \mathcal{K}_1(i\xi))\beta^k \\ (-\lambda_- + \mathcal{K}_2(i\xi))\beta^{k+1} = (-\lambda_+ + \mathcal{K}_2(i\xi))\alpha^k \end{cases}$$

or using a matrix notation

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}^{k+1} = \Delta(i\xi) \begin{pmatrix} \alpha \\ \beta \end{pmatrix}^k$$

and the convergence matrix $\Delta = \Delta(i\xi)$ is

$$\Delta = \begin{pmatrix} 0 & \dfrac{\lambda_-(i\xi) + \mathcal{K}_1(i\xi)}{\lambda_+(i\xi) + \mathcal{K}_1(i\xi)} \\ \dfrac{\lambda_+(i\xi) - \mathcal{K}_2(i\xi)}{\lambda_-(i\xi) - \mathcal{K}_2(i\xi)} & 0 \end{pmatrix}$$

The (α^k, β^k) sequence converges if and only if it converges to 0 , thus if and only if the spectral radius of Δ is less than 1. In this case, the two eigenvalues of the matrix Δ are

$$\pm \left(\frac{\lambda_-(i\xi) + \mathcal{K}_1(i\xi)}{\lambda_+(i\xi) + \mathcal{K}_1(i\xi)} \times \frac{\lambda_+(i\xi) - \mathcal{K}_2(i\xi)}{\lambda_-(i\xi) - \mathcal{K}_2(i\xi)} \right)^{\frac{1}{2}}$$

Again it is to be seen that if $\lambda_- + l_1 = 0$ and $\lambda_+ - l_2 = 0$, the convergence matrix is null and the algorithm converges in 2 iterations. When approximated operators are used, the convergence rate is given by:

$$\delta(i\xi) = \left(\frac{\lambda_-(i\xi) + l_1(i\xi)}{\lambda_+(i\xi) + l_1(i\xi)} \times \frac{\lambda_+(i\xi) - l_2(i\xi)}{\lambda_-(i\xi) - l_2(i\xi)} \right)^{\frac{1}{2}}$$

On the face of it, operators l_1 et l_2 are independent. However as

$$\forall \xi \in \mathbb{R}, \lambda_+(i\xi) + \lambda_-(i\xi) = \frac{a}{\nu}$$

and as operators l_1 and l_2 approximate $-\lambda_-$ and λ_+, they can be chosen such that the following relation should be satisfied

$$\mathcal{K}_2 - \mathcal{K}_1 = \frac{a}{\nu}$$

In this case

$$\delta(i\xi) = \frac{\lambda_+(i\xi) - \mathcal{K}_2(i\xi)}{\lambda_-(i\xi) - \mathcal{K}_2(i\xi)}$$

In [4], optimization of the convergence rate was proved to be successful, for operators l_j that were polynomials of degree less than 2 in ξ. Here, only Robin type boundary conditions will be looked into, that is operators of the form $\mathcal{K}_2 \equiv p_{opt}$ such that p_{opt} realises

$$\inf_{p\in\mathbb{R}^+} \sup_{\xi\in[\xi_0,\xi_1]} \left| \frac{\lambda_+(i\xi) - p}{\lambda_(i\xi) - -p} \right| \tag{8}$$

where ξ_0 is related to the length of the Y interval, $\xi_0 = \frac{\pi}{L}$, and ξ_1 to the Y discretisation, $\xi_1 = \frac{\pi}{\Delta Y}$.

There is a unique p_{opt} minimizing the convergence rate, and it is characterized by $\delta(\xi_0) = \delta(\xi_1)$, [6].

p_{opt} thus computed is a function of the equation coefficients and the coupled problem (4) is solved with

$$\begin{cases} \mathcal{B}_1 = \dfrac{\partial}{\partial n_1} - \dfrac{a}{\nu} + p_{opt} \\ \mathcal{B}_2 = \dfrac{\partial}{\partial n_2} + p_{opt} \end{cases} \tag{9}$$

2.2 Evolution equations

A different approach is now proposed: the evolution equation itself is solved with a domain decomposition technique [2] allowing for different time steps in each subdomain.

Let us resume the evolution equation

$$\mathcal{P}U \equiv \frac{\partial U}{\partial t} + \mathbf{a} \cdot \nabla U - \nu \Delta U = 0 \text{ in } \Omega \times (0,T),$$

associated with the initial data $U(\cdot,0) = U^0$ in Ω at time 0 and the Dirichlet boundary condition $U = 0$ on $\Gamma = \partial\Omega$. A global Schwarz algorithm in time can be developed in the two subdomains $\Omega_1 \times (0,T)$, et $\Omega_2 \times (0,T)$,

$$\begin{cases} \mathcal{P}v^{k+1} = 0, \quad \text{in } \Omega_1 \times (0,T) \\ v^{k+1}(.,0) = U^{(0)} \text{ in } \Omega_1 \\ v^{k+1} = 0 \text{ on } \Gamma_1 \times (0,T) \\ \mathcal{B}_1 v^{k+1} = \mathcal{B}_1 w^k \text{ on } \Gamma_{12} \times (0,T) \end{cases} \tag{10}$$

$$\begin{cases} \mathcal{P}w^{k+1} = 0, \quad \text{in } \Omega_2 \times (0,T) \\ w(.,0) = W^{(0)} \text{ in } \Omega_1 \\ w^{k+1} = 0 \text{ on } \Gamma_2 \times (0,T) \\ \mathcal{B}_2 w^{k+1} = \mathcal{B}_2 v^k \text{ on } \Gamma_{12} \times (0,T) \end{cases} \tag{11}$$

The error $(W^k = w^k - u, V^k = v^k - u)$ is solution of the same problem with null initial conditions. Hence it is zero at step 2 if the boundary operators $\mathcal{B}_1$ and $\mathcal{B}_2$ are transparent operators $\mathcal{B}_1^{T,ev}$ and $\mathcal{B}_2^{T,ev}$ defined as above for evolution problems: let g be defined on $\Gamma_{12} \times (0,T)$ and define the problem

$$\begin{cases} \mathcal{P}w = 0 \text{ in } \Omega_2 \times (0,T) \\ w(.,0) = 0 \text{ in } \Omega_2 \\ w = 0 \text{ on } \Gamma_2 \times (0,T) \\ w = g \text{ on } \Gamma_{12} \times (0,T) \end{cases}$$

The Dirichlet-Neumann operator for Ω_2, $\mathcal{K}_1^{T,ev}$ can be defined as

$$\mathcal{K}_1^{T,ev} g = \frac{\partial w}{\partial n_2},$$

Thus any solution of the boundary problem in Ω_2

$$\begin{cases} \mathcal{P}w = 0 \text{ in } \Omega_2 \times (0,T) \\ w(.,0) = 0 \text{ in } \Omega_2 \\ w = 0 \text{ on } \Gamma_2 \times (0,T) \end{cases}$$

also satisfies equation on $\Gamma_{12} \times (0,T)$

$$\frac{\partial w}{\partial n_2} - \mathcal{K}_1^{T,ev} w = 0$$

Similarly for operator $\mathcal{K}_2^{T,ev}$, the problem

$$\begin{cases} \mathcal{P}v = 0 \text{ in } \Omega_1 \times (0,T) \\ v = 0 \text{ on } \Gamma_1 \times (0,T) \\ v(.,0) = 0 \text{ in } \Omega_1 \\ v = g \text{ on } \Gamma_{12} \times (0,T) \end{cases}$$

The Dirichlet-Neumann operator for Ω_1, $\mathcal{K}_2^{T,ev}$ is defined as

$$\mathcal{K}_2^{T,ev} g = \frac{\partial v}{\partial n_1}$$

Any solution of the boundary problem in Ω_1

$$\begin{cases} \mathcal{P}v = 0 \text{ in } \Omega_1 \times (0,T) \\ v(.,0) = 0 \text{ in } \Omega_1 \\ v = 0 \text{ on } \Gamma_1 \times (0,T) \end{cases}$$

satisfies equation on Γ_{12}

$$\frac{\partial v}{\partial n_1} - \mathcal{K}_2^{T,ev} v = 0.$$

Operators

$$\mathcal{B}_1^{T,ev} = \frac{\partial}{\partial n_1} + \mathcal{K}_1^{T,ev}$$
$$\mathcal{B}_2^{T,ev} = \frac{\partial}{\partial n_2} + \mathcal{K}_2^{T,ev}$$

are **exact transmission operators** , also named **transparent operators**.

As in the steady case , they can be locally identified through a Fourier transform (parameter ξ) in the transverse direction and a Laplace transform in time (parameter s). $\eta = \Re(s) \geq \eta_0 > 0$ as solutions are bounded in time.

More precisely , the partial Fourier transform in relation to Y and the Laplace transform in relation to t is defined as :

$$\hat{u}(x, \xi, s) = \frac{1}{2\pi} \int_{\mathbb{R}} \int_{\mathbb{R}_+} u(x, Y, t) e^{-i\xi Y - st} dY\, dt, s = \eta + i\omega, \eta \geq \eta_0$$

The Fourier transform of v is solution of an ODE in x, where k and s appear as parameters.

$$\begin{cases} -\nu\dfrac{\partial^2\hat{V}}{\partial X^2} + \boldsymbol{a}\cdot\boldsymbol{n}\dfrac{\partial\hat{V}}{\partial X} + (s + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2)\hat{V} = 0, x < 0, \\ \hat{V} = \hat{g} \text{ at } x = 0, \end{cases} \tag{12}$$

The characteristic polynomial associated to this ODE is :

$$P(i\xi, s; \lambda) = -\nu\lambda^2 + \boldsymbol{a}\cdot\boldsymbol{n}\lambda + (s + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2)$$

Its discriminant is:

$$\boldsymbol{a}\cdot\boldsymbol{n}^2 + 4\nu(s + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2) \tag{13}$$

The real part of (13) is strictly positive. Let us write ρ^{ev} the square root of (13) with strictly positive real part

$$(\rho^{ev})^2(i\xi, s) = a^2 + 4\nu(s + i\boldsymbol{a}\cdot\boldsymbol{\tau}\xi + \nu\xi^2), \quad \Re\rho > 0.$$

In this case $\Re\rho > a$. Polynomial $P(i\xi, s; \cdot)$ has two complex roots

$$\lambda_{\pm}^{ev}(i\xi, s) = \frac{\boldsymbol{a}\cdot\boldsymbol{n} \pm \rho}{2\nu}$$

which satisfy $\Re\lambda_{+}^{ev}(i\xi, s) > 0$ and $\Re\lambda_{-}^{ev}(i\xi, s) < 0$.

As in the case of absorbing boundary conditions in [3] , there exists a function $\alpha(\xi)$, locally integrable, such that $\hat{v} = \alpha e^{\lambda_1^{ev} x}$, and v satisfies

$$\frac{\partial\hat{v}}{\partial x} = \lambda_{+}^{ev}\hat{V} \text{ for } x < 0.$$

The operator $\mathcal{K}_2^{T,ev}$ on the boundary is given by

$$\mathcal{K}_2^{T,ev}(\frac{\partial}{\partial Y}) = \lambda_{+}^{ev}(\frac{\partial}{\partial Y})$$

In the same fashion, operator $\mathcal{K}_1^{T,ev}$ is defined by

$$\mathcal{K}_1^{T,ev}(\frac{\partial}{\partial Y}) = -\lambda_{-}^{ev}(\frac{\partial}{\partial y})$$

Operators $\mathcal{B}_1^{T,ev}$ and $\mathcal{B}_2^{T,ev}$ are thus

$$\begin{cases} \mathcal{B}_1^{T,ev} = \dfrac{\partial}{\partial X} + \mathcal{K}_1^{T,ev}(\dfrac{\partial}{\partial Y}), \\ \mathcal{B}_2^{T,ev} = -\dfrac{\partial}{\partial X} + \mathcal{K}_2^{T,ev}(\dfrac{\partial}{\partial Y}) \end{cases}$$

As in the steady case, Robin type approximations of the transmission conditions are introduced in algorithm (10),(11), see [1]

$$\begin{cases} \mathcal{B}_1 = \dfrac{\partial}{\partial n_1} - \dfrac{a}{\nu} + p_{opt} \\ \mathcal{B}_2 = \dfrac{\partial}{\partial n_2} + p_{opt} \end{cases} \tag{14}$$

where p_{opt} minimizes the convergence ratio.

$$\delta(i\xi, s) = \frac{\lambda_+(i\xi, s) - p}{\lambda_-(i\xi, s) - p} \quad :$$

$$\inf_{p \in \mathbb{R}^+} \sup_{\xi \in [\xi_0, \xi_1], \eta \geq \eta_0, \omega \in [\omega_0, \omega_1]} \left| \frac{\lambda_+(i\xi, s) - p}{\lambda_-(i\xi, s) - p} \right|$$

where ξ_0 is related to the length of the Y interval, $\xi_0 = \frac{\pi}{L}$, and ξ_1 to the Y discretisation, $\xi_1 = \frac{\pi}{\Delta Y}$. In the same way, $\omega_0 = \frac{\pi}{T}$ and $\omega_1 = \frac{\pi}{\Delta T}$ while η_0 is chosen arbitrarily small.

For analyticity reasons , it is sufficient to make sure of an upper bound for $\eta = \eta_0$ and p_{opt} is computed to realize

$$\inf_{p \in \mathbb{R}^+} \sup_{\xi \in [\xi_0, \xi_1], \eta = \eta_0, \omega \in [\omega_0, \omega_1]} \left| \frac{\lambda_+(i\xi, s) - p}{\lambda_-(i\xi, s) - p} \right| \tag{15}$$

This ends the presentation of the mathematical coupling. We now proceed to introduce the physical coupling technology.

3 Metacomputing

Development of software and hardware technologies has lead to the new paradigm of **distributed** or **meta computing**. Object Oriented programming, client server model and software bus paradigm will be used to set up a new class of applications which might involve several simulation codes running on different sites and multiple interacting users. One key point here is to be able to deal with heterogeneity of computing languages, operating systems and hardware platforms.

In this framework applications are seen as "object-servers" that interact transparently through the network. We introduce briefly some useful programming and run time models for distributed applications.

3.1 Programming and run time models

The first step is to encapsulate our applications in "objects". An object has attributes -mainly the encapsulated data- which define its state and a behavior described by its methods -or procedures-. In a general object oriented programming model one sends "messages" to objects to request execution of a local method. Hence, as described in the last sentence, typical objects are "servers". They are able to perform a given task, i.e. *"provide a service"* upon reception of a message, i.e. *"upon request from a client"*.

It is rather straightforward to design a client-server architecture from existing software. Just think of the usual "main program" of the application as a client code requesting services from a "library part". The functions called in this main program and the data they are working on belong to the server side, hence parts of the server can be viewed as a library. In a standard stand-alone application everything is on the same platform. In a distributed context, the caller -client side- and the library -server side- may reside on separate platforms.

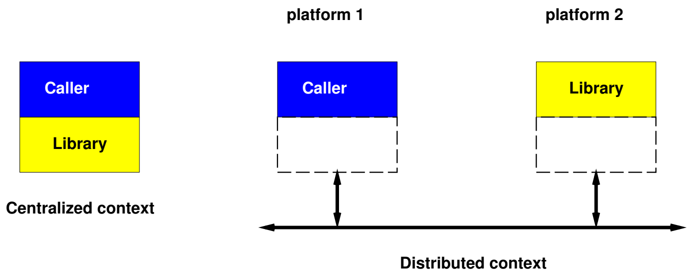


Figure 3: A client/server application (1)

Just like hardware components are connected by a bus in a computer, a software bus is a model where applications are seen as components that can be "plugged" on the bus. All plugged components may interact without modifications of the existing ones. Figure 4 shows a "software bus" vision of a distributed system involving several servers.

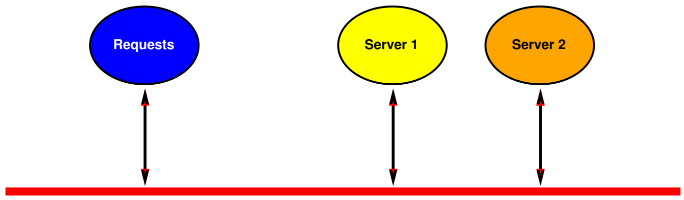


Figure 4: A software Bus

When Object Oriented Programming is used to design a client-Server architecture, we will use the term "distributed objects" and will refer to "objects living on the server side" and "objects living on the client side". In this framework, the run time model is defined as objects interacting by exchanging messages. Those objects may live anywhere in the distributed system. This is transparent to the user. Here we will use `C++` for object oriented programming but an interesting alternative is to use the `Java` programming language [8, 9] for large numerical simulations [10, 11]. The Remote Method Invocation (`RMI`) mechanism introduced in `Java` [12] allows one to design distributed objects applications.

Most recent software environments now address the "**Computing Grid**" where project developments integrate geographically distributed resources shared by groups of users [13, 14]. The challenge for many research projects such as `Globus` [15, 16] or `Legion` [17, 18] is to build powerful and "user friendly" hardware and software environments that provide uniform and network transparent access to those resources (computing power, information, applications, data bases, ...).

Our project is obviously more limited but we aim at designing applications as a set of software components that can be reused within any general framework.

3.2 Corba

Corba [19, 20, 21] stands for *"Common Object Request Broker Architecture"*. It is an attempt to define a standard for programming distributed applications with the three models discussed above: object oriented; client-server and software bus. As stated before this is "Distributed Objects" but objects will now communicate through the Corba software bus.

Corba is defined by the OMG [22] which is a consortium of users and hardware or software vendors. It is an open standard in the sense that all members of the OMG consortium agree to it and may examine new proposals. The OMG does not provide any implementation of the standard.

Let us just cite some main goals of Corba which are particularly relevant to the context of distributed numerical simulation:

- link with existing programming languages (C, C++, Java, ...);
- automatic and transparent activation of objects; transparency of method invocation: "local" or "remote";
- self descriptive system. Requests can be made at compile time -statically- or built at run-time -dynamically-;
- interoperability of Corba buses -i.e. of various implementations- through well defined protocols.

The user's view of a Corba system is quite simple, see figure 5. The server may be located anywhere but is accessed through an interface. The client issues requests i.e. method invocations, on a "proxy". Corba (the request broker, the bus, ...) takes care of the requests and potential results. An implementation of a standard like Corba is called a **middleware** as it makes the link between the applications (software level) and computers and their operating systems (hardware level).

Figure 5: Corba, a user's view

The Corba standard includes the definition of an Interface Description Language called IDL. Using IDL allows us to be independent from the programming language used *in fine*. IDL is not a programming language since it is just used to describe interfaces.

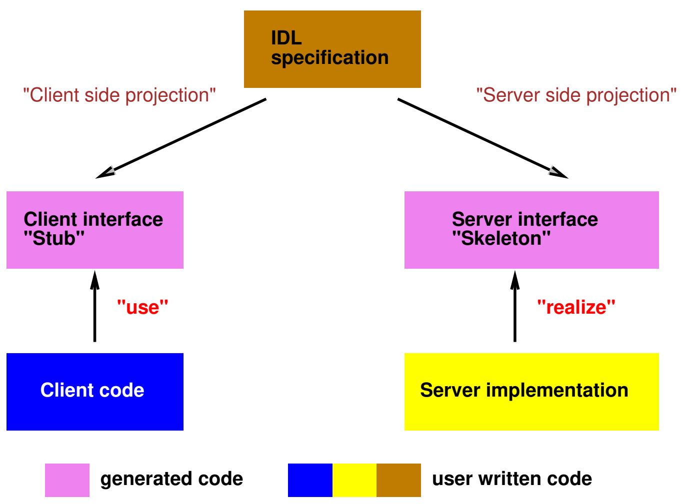


Figure 6: Structure of a "`Corba` software"

The `C++` language [23] is now widely used for large scale simulations [24]. For `C++` users, the `IDL` specification almost matches the specifications in the `public` part of a `class` construct. The `IDL` compiler makes "projections" of the specifications onto the desired programming languages for the client and for the server. The projection on the client side is called a `stub`: it is the interface used by the server. The projection on the server side is called a `skeleton`: here the operations have to be implemented.

If an object oriented language such as `C++` is used for both projections the whole process is easy to handle through inheritance. Here, the `IDL` compiler generates an interface (i.e. specification part of a `C++` class) from the `IDL` description. Stub and Skeleton are generated as classes derived from this interface by inheritance and finally the server implementation inherits from the generated skeleton as in figure 6.

Going back to figure 3 we see in figure 7 that we have filled the dashed boxes: the stub and the skeleton are the components which allow us to distribute the application. Local and remote calls now have the same syntax and the same semantic.

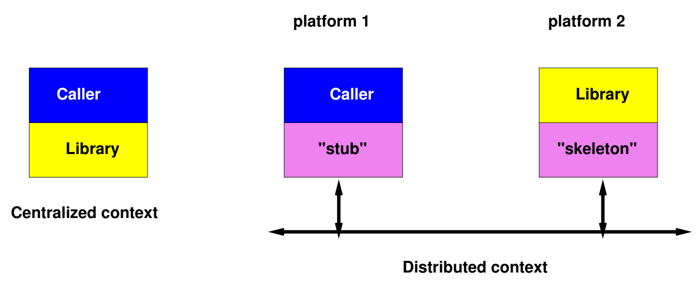


Figure 7: A client/server application (2)

3.3 Conclusions

We have shown how an application can be developed as an "object server" and then can be run through remote control. Obviously this approach stands for new applications, but legacy software (typically `Fortran` codes) can easily be "encapsulated" within an object framework (e.g. a `C++` class) and turned into such a server without major restructuring.

As a consequence software developed inside this framework does not need to be ported locally to be accessible. It might even be a commercial application made available by an "Application Service Provider" (`ASP`).

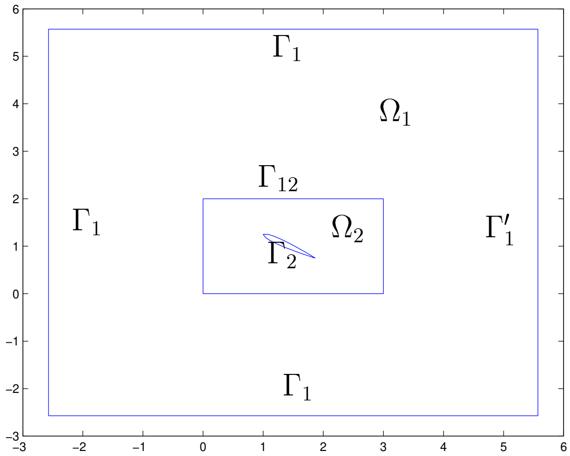
Once a server is developed, it is ready to be used as a component within a distributed application. Of courses new services will be needed for the components to communicate. It remains now to test the feasibility of this technology on a domain decomposition test case.

4 Application

An important aspect of domain decomposition is that it allows for geometric simplification with different discretisations along with various modelisations. Complex geometrical elements are time consuming and should be as few as possible. A typical problem is meshing with smoothness the connection of a wing to the fuselage, or any complex body within a large regular domain. Our test case concerns the coupling of finite elements with finite differences discretisations of a same equation, relevant to this problem .

4.1 The test case

The equation is the two-dimensional convection diffusion of a scalar (temperature), see equation (1) given in section 2 with $\nu = 0.01$ and $\mathbf{a} = (1, 0)$, describing a heat bubble hitting an up-going Naca airfoil. The computing domain consists of two subdomains, the first around the airfoil using triangles for a better definition of the profile, the second domain discretising the far field with regular rectangles. External boundary conditions are imposed: Dirichlet values on the upstream ($u = v_1$), top ($u = 0$), bottom sides ($u = 0$) and an homogeneous Neumann boundary condition on the downstream side and the airfoil, see figure 8.



Γ_1 is a Dirichlet boundary,
Γ_1' is a Neumann boundary ,
Γ_2 is the airfoil with Neumann boundary,
Ω_1 is the Finite Differences domain,
Ω_2 is the Finite Elements domain,
Γ_{12} is the interface boundary between Finite Elements and Finite Differences

Figure 8: non-overlapping 2 domains decomposition

4.2 Numerical algorithms

Two independent different codes (`EF`, `DF`) were coupled, see figures 9 and 10. Both solve the two-dimensional convection diffusion of the scalar u in equation (1) with the same time scheme (Crank-Nicolson).

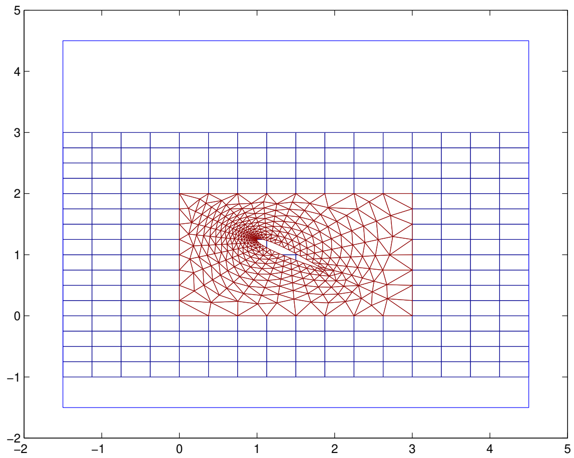


Figure 9: General mesh

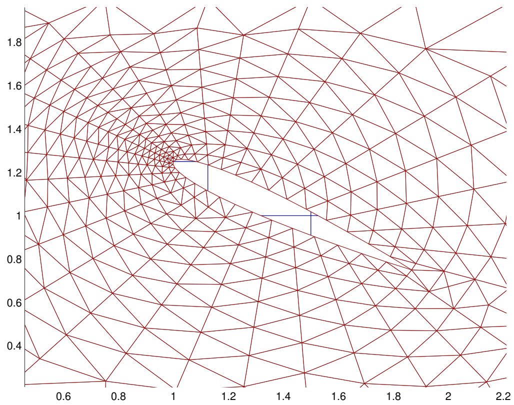


Figure 10: Closeup of the airfoil

`EF` space discretisation uses P1 finite elements, on the triangular mesh close to the airfoil, the implicit part in the time scheme is solved using a Block Lower Upper decomposition.

`DF` space discretisation is based on second order finite differences on a cartesian grid, the implicit part in the time scheme is solved iteratively with a GMRES solver . The Finite Difference grid meshes in one block the whole domain including the airfoil, points

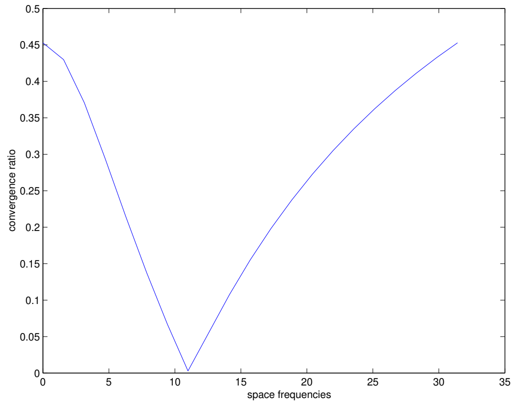


Figure 11: Steady case

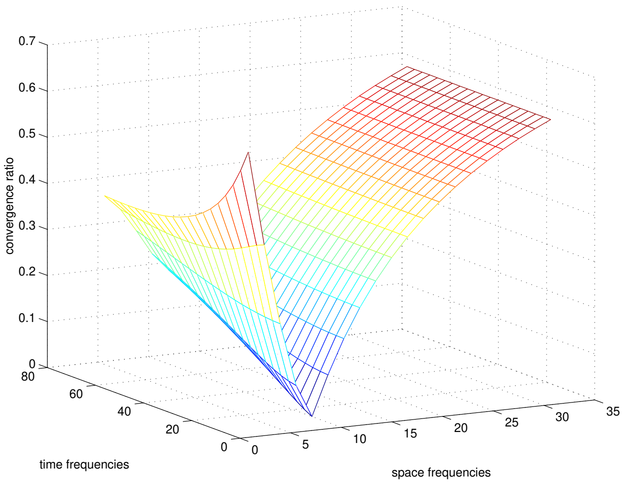


Figure 12: Unsteady case

situated on the grid within the inner Finite Element domain are set to zero and boundary conditions are imposed on the interface with the finite element domain.

These two codes are coupled using the Schwarz iterative procedure presented above, with Robin type transmission operators $\frac{\partial u}{\partial n} + \alpha_i u$ on the interface.

4.2.1 Computation of the Robin coefficients

As shown in section 2 , the $(\alpha_i)_{i=1,2}$ are chosen so as to minimise the convergence rate of the Schwarz algorithm and are given by $\alpha_2 = \mathbf{a}.n/\nu - \alpha_1$ and $\alpha_1 = p$ such that $\inf_{p \in \mathbb{R}^+} \sup_{\xi \in [\xi_0, \xi_1]} \left| \frac{\lambda_+ - p}{\lambda_- - p} \right|$ where ξ_0 is related to the length of the y interval, $\xi_0 = \frac{\pi}{L}$, and ξ_1 to the y discretisation, $\xi_1 = \frac{\pi}{\Delta y}$.

These coefficients were numerically computed using a an iterative zooming technique. p is first determined (as p_0) in a initial interval $[0, p_{max}]$ meshed with N points. Then another $p = p_1$ is searched in $[p_0 - \epsilon, p_0 + \epsilon]$ meshed with N points. Search is ended when in the case of steady coupling $\frac{\lambda_+(kmin) - p}{\lambda_-(kmin) - p} = \frac{\lambda_+(kmax) - p}{\lambda_-(kmax) - p}$ [6], and in the unsteady case when $p_i - p_{i+1} < 10^{-12}$ (kmin and kmax are minimum and maximum frequencies). Three or four iterations were sufficient for N=1000.

In figures 11 and 12, convergence rate for the Robin value minimising (8) and (15) is shown versus space frequencies in the steady case, and versus space and time frequencies in the unsteady case.

4.2.2 Steady coupling

In the steady case the algorithm is the following: at each time step with index n, and for each Schwarz iteration with index k, the following problems are solved:

$$
\begin{cases}
\mathcal{L}_1 v_k^{n+1} = F_1, \quad \text{ in } \Omega_1 \\
v_k^{n+1} = v_1 \text{ on } \Gamma_1 \\
\dfrac{\partial v_k^{n+1}}{\partial n} = 0 \text{ on } \Gamma_1' \\
\mathcal{B}_1 v_k^{n+1} = \mathcal{B}_1 w_k^n \text{ on } \Gamma_{12}
\end{cases}
\tag{16}
$$

where $\mathcal{L}_1$ is the second order finite difference discretisation of $(Id + dt/2\mathbf{a}\nabla - \nu\triangle)$ and $F_1 = (Id - dt/2\mathbf{a}\nabla - \nu\triangle)v_k^n$ and the boundary condition is imposed explicitly

$$
\begin{cases}
\mathcal{L}_2 w_{k+1}^{n+1} = F_2, \quad \text{ in } \Omega_2 \\
\dfrac{\partial w_k^{n+1}}{\partial n} = 0 \text{ on } \Gamma_2 \\
\mathcal{B}_2 w_k^{n+1} = \mathcal{B}_2 v_{k+1}^n \text{ on } \Gamma_{12}
\end{cases}
\tag{17}
$$

These second set of equations are discretised using finite element integration in space and finite differences in time. They become

$$
\begin{array}{l}
\displaystyle\int_{\Gamma_{12}} \alpha_2 w_{k+1}^{n+1} v d\gamma + \\
\displaystyle\quad \int_{\Omega_2} (w_{k+1}^{n+1} v + dt/2(\mathbf{a}\nabla) w_{k+1}^{n+1} v + \nu dt/2 \nabla w_{k+1}^{n+1} \nabla v) dx = \\
\displaystyle\int_{\Omega_2} \left\{ w_k^n v - [(dt/2\mathbf{a}\nabla) w_k^n v + \nu dt/2 \nabla w_k^n \nabla v] \right\} dx \\
\displaystyle + dt/2 \int_{\Gamma_{12}} (\mathcal{B}_2 v_{k+1}^n + \frac{\partial w_k^n}{\partial n}) v d\gamma
\end{array}
\tag{18}
$$

where v is any element of the P1 basis and $\mathcal{B}_2 u = \alpha_2 u + \dfrac{\partial u}{\partial n}$

The Schwarz iterative procedure can be seen as a 4-block Gauss-Seidel method where two of the blocks contain the discretisation of the subdomains $(A_{ii})_{i=1,2}$ and the other two the coupling along the artificial boundaries $(A_{ij})_{i,j=1,2}$

$$
\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}
\tag{19}
$$

At this point, two remarks can be made . 1) It is well known that any Krylov iterative technique is an improvement in terms of number of iterations over the Gauss-Seidel solver. 2) Krylov techniques imply storing residual vectors of size the number of points in all the domains, and can be quite storage consuming if applied directly to (19). Therefore the problem is transformed into a linear problem on the interface Γ_{12} .
For simplicity , we shall note $v_{k+1}^{n+1} = v$ and $w_{k+1}^{n+1} = w$
$v = v_0 + \bar{v}$ and $w = w_0 + \bar{w}$ where v_0 and w_0 are solutions of the problems

$$
\begin{cases}
\mathcal{L}_1 v_0 = F_1, \quad \text{ in } \Omega_1 \\
v_0 = v_1 \text{ on } \Gamma_1 \\
\dfrac{\partial v_0}{\partial n} = 0 \text{ on } \Gamma_1' \\
\mathcal{B}_1 v_0 = 0 \text{ on } \Gamma_{12}
\end{cases}
\tag{20}
$$

$$
\begin{cases}
\mathcal{L}_2 w_0 = F_2, \quad \text{ in } \Omega_2 \\
\dfrac{\partial w_0}{\partial n} = 0 \text{ on } \Gamma_2 \\
\mathcal{B}_2 w_0 = 0 \text{ on } \Gamma_{12}
\end{cases}
\tag{21}
$$

$\bar{v}$ and $\bar{w}$ are now solutions of

$$
\begin{cases}
\mathcal{L}_1 \bar{v} = 0, \quad \text{ in } \Omega_1 \\
\bar{v} = 0 \text{ on } \Gamma_1 \\
\dfrac{\partial \bar{v}}{\partial n} = 0 \text{ on } \Gamma_1' \\
\mathcal{B}_1 \bar{v} = \mathcal{B}_1 \bar{w} \text{ on } \Gamma_{12}
\end{cases}
\tag{22}
$$

$$
\begin{cases}
\mathcal{L}_2 \bar{w} = 0, \quad \text{ in } \Omega_2 \\
\dfrac{\partial \bar{w}}{\partial n} = 0 \text{ on } \Gamma_2 \\
\mathcal{B}_2 \bar{w} = \mathcal{B}_2 \bar{v} \text{ on } \Gamma_{12}
\end{cases}
\tag{23}
$$

Both (22) and (23) are linear problems with respect to the boundary conditions γ_1, γ_2 on Γ_{12} .

Determining the coupled values (v,w) can now be seen as solving the following 2-field linear operator $G(\gamma_1, \gamma_2) = -(\mathcal{B}_1(v_0 - w_0), \mathcal{B}_2(v_0 - w_0))$ where $G(\gamma_1, \gamma_2) = (\mathcal{B}_1(\bar{v} - \bar{w}), \mathcal{B}_2(\bar{v} - \bar{w}))$

A Krylov technique applied to this problem will have residuals the size of the interface which is small compared to the whole domain.

This iterative solver has to be applied at each time step in the case of the steady coupling. A natural preconditioner is to project the initial solution v_0^{n+1} on former search vectors defined when computing v^n. This is easily implemented within the FOM solver (Full Orthogonalisation Method).

Remark: $\mathcal{B}_1(\bar{w})$ is computed with the `EF` code in the following way: imposing $\gamma_2 = \alpha_2 \bar{w} + \dfrac{\partial \bar{w}}{\partial n}$ provides after resolution $\bar{w}$, thus $\dfrac{\partial \bar{w}}{\partial n} = \gamma_2 - \alpha_2 \bar{w}$ and $\mathcal{B}_1(\bar{w}) = (\alpha_1 - \alpha_2)\bar{w} + \gamma_2$
$\mathcal{B}_2(\bar{v})$ is computed using finite differences.

4.2.3 Unsteady coupling

Having reduced the problem to one on the interface, the unsteady coupling can be seen as $G(\gamma_1(t), \gamma_2(t)) = (\mathcal{B}_1(\bar{v}(t) - \bar{w}(t)), \mathcal{B}_2(\bar{v}(t) - \bar{w}(t)))$ where t is a vector $(t_1, ..., t_N)$ representing time discretisation of $[T_1, T_2]$ and residuals have size the interface dimension times N. Thus the unsteady coupling algorithm is identical to the steady one. The only difference is that instead of computing at each time step $G(\gamma_1(t_i), \gamma_2(t_i))$ implying small communications, the unsteady algorithm computes directly the block $G(\gamma_1(t), \gamma_2(t))$ every N time steps, thus a greater amount of data exchanged but less often.
Schematically:

the steady algorithm is:

```
do time iterations 1:N
 do Krylov iterations
  residual vectors =
  size of interface
```

the unsteady algorithm is:

```
do Krylov iterations
 do time iterations 1:N
  residual vectors =
  size of interface x N
```

4.3 Convergence results on the test case

Steady case
On figure (13) convergence with and without preconditioning is presented for two time steps, situated when the heat blob is entering the finite element domain.

Unsteady case
As can be seen on figure 14, the unsteady domain decomposition technique over a time window of 10 time steps converges in terms of number of iterations similarly to 10 preconditioned steady time steps. The unsteady coupling acts as a global preconditioner.

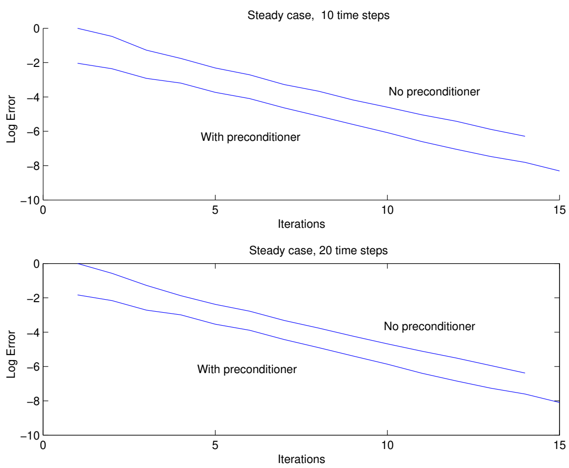


Figure 13: Effect of the preconditionner

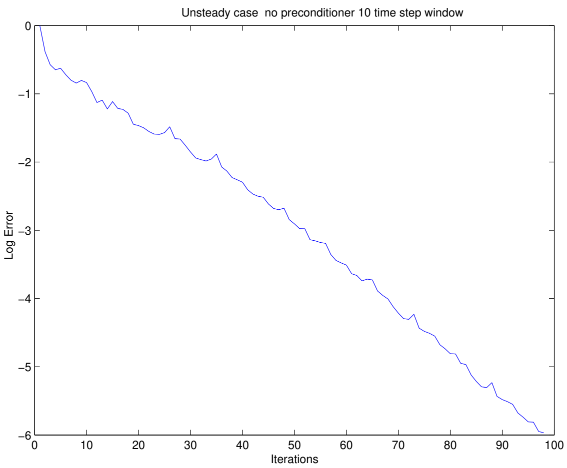


Figure 14: Unsteady case

5 Implementation with Corba

In this section, we describe how we implemented the coupling of two `Fortran` numerical applications. Legacy softwares are encapsulated within `C++` classes, “turned into servers” and immersed in a `Corba` object oriented client-server environment. Finally we can implement the coupling without porting and major restructuring of the original applications.

5.1 From `Fortran` codes to object servers

For our project two applications are involved, they *a priori* need to be ported locally and then coupled. Porting an application always leads to many annoying problems and one never knows what to do with the computations which are specific to the coupling.

To overcome these problems we encapsulate the "user's relevant" components of an application so that they become methods of an "object server". This is done by designing a `C++` class as described in figure 15

Following the methodology presented in 3.1, the methods are the services to be provided by the `appli` object server. Those methods match the subroutines called in the main `Fortran` program and a few simple requests for setting initial data, etc.... The variables declared in the main `Fortran` program will have their counterpart encapsulated in the private part of a `appli` object. A client code will look like the body of the main `Fortran` program and invoke services on a reference to a `appli` object.

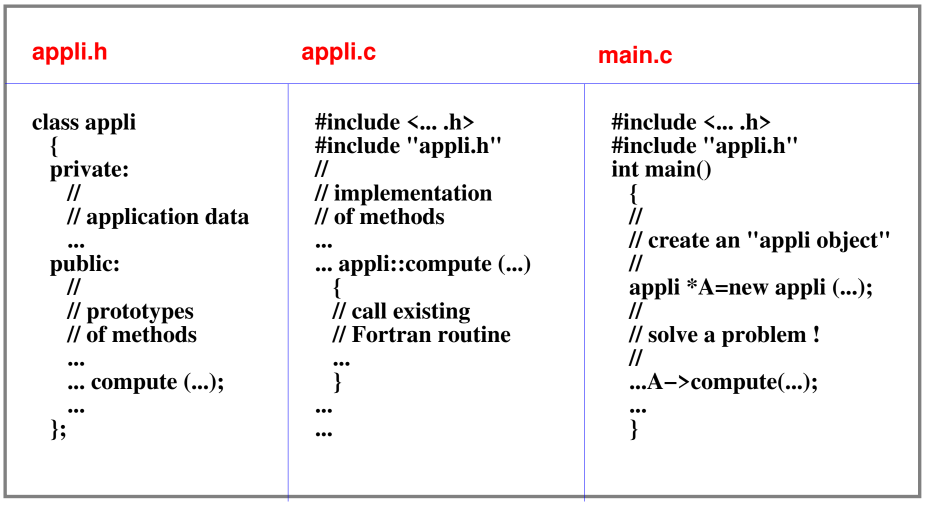


Figure 15: `C++` encapsulation

Then we derive a `IDL` specification from the `C++` description of the class and the `IDL` compiler is used to generate `C++` stub and skeleton used in figure 7. Now, the implementation will inherit from the generated skeleton and will implement the functionalities of the object server for the given application as in figure 6: it will be simply derived from the previous `C++` implementation of `appli`.

The client code will use the stub interface as the specification for the server and will also simply be derived from the `C++` main program. It will invoke services on a reference to a remote `appli` object through the "proxy" as shown in figure 5.

If we compare with the initial "monolithic" application, we are now able to solve the same problem by running the code through remote control, i.e. by sending requests to an object server. In this distributed context any user may write a client code looking like his usual "`main`" program and run it **locally**. The application will connect to a remote server which will carry over computations and solve the problem. No porting is to be done. Furthermore, we did not really have to "enter" the code: we just analysed the main program, identified application data and exhibited the relevant functionalities. An important point is that we used codes which were not originally developped for distributed simulations. By wrapping the existing subroutines in our servers we salvaged all the scientific knowledge imbedded in these applications.

`Corba` allows us to achieve interoperability. Our software construct is independent from the programming languages we use and from the operating systems and hardware the client and the server are running on.

5.2 Design of the coupling

As we are able to run the codes through remote control, it is now possible to undertake the coupling of applications provided we make available the basic functionalities needed to express interactions between them **in a client code**.

From a technical point of view, many software architectures can be foreseen for this system, where all additional computations are taken over by the client or by another secondary module. Two typical software architectures are shown in figure 16. In the first diagram the client controls the two applications and does all the additional work. In the second diagram, the client just controls the processes while additional work is delegated to a "coupling module".

If we were able to build a "real software bus" (*c.f.* figure 4), *i.e.* work with a software component model, any software architecture for the system would be transparent to the user. Unfortunately, this is not really the case with current implementations of `Corba` which do not provide enough abstraction for that: clearly here codes for servers would be different **but** encapsulations would not be affected. For this test case we will keep to the first diagram. The new `Corba` component model (`CCM`) recently released by the `OMG` will provide more flexibility in the design of distributed applications [25].

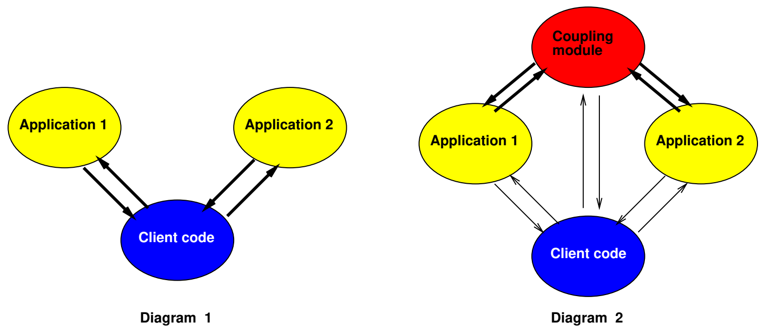


Figure 16: Software components for coupling

To undertake this coupling, *i.e.* to implement the algorithms developed in 4 we need to add new functionalities -services- to our servers:

- identify boundaries;
- extract the values of a given unknown on a given boundary;

- extract the values of time or space derivatives of a given unknown on a given boundary;
- modify the values of a given unknown on a given boundary;
- access various scalar values (coefficients for optimal convergence, ...).

Our servers need to be upgraded as they must include the new "coupling functionalities" but these features are rather simple, "generic" and easy to implement in Fortran in the initial code. Thanks to those functionalities, data can be transferred transparently from one application to any other hence interpolations and additional computations can be performed by any module.

The coupling will be entirely expressed in the new client code in which we will monitor both applications and perform the additional computations.

Just to give a flavour of the corresponding IDL specification, additional functionalities for both servers will look like:

```
// ...
// here, "fields" are arrays of double
// borders are identified by "border_id"
// ...
// access values on a boundary
//
void get_u     (in  long border_id, out field Uo );
void get_du_dn (in  long border_id, out field Uo );
//
// get coeficients for optimal convergence
//
void get_alpha (out double alpha                );
void get_beta  (out double beta                 );
//
// update values on a boundary
//
void update_u  (in  long border_id, in  field Ui );
// ...
```

Figure 17: IDL upgrades for servers

5.3 Handling parallelism

Obviously, in this new client code, many things can be done in parallel. For example, each server can be monitored inside a **thread** (with the suitable synchronizations...)

to implement the inter-application parallelism level in the client code. Multi threaded implementations of `Corba` generally offer a `‘‘java-like’’` abstraction of threads through a `C++` thread class. Also in `C++`, multiple inheritance allows to take care of thread parallelism in the object server itself.

On the other hand, intra application parallelism must be handled too. Here the original codes are parallel applications based on the `MPI` library [26, 27]. At the `Corba` level -from the client point of view- we do not know that they are parallel applications and cannot take benefit from it. The consequence is that, at some step, exchanges are not performed among processes of the parallel applications but sequentialization has to occur since only one process of each application can dialog through `Corba` as shown in figure 18. Another problem in figure 18 is that `MPI` applications are no longer `SPMD` codes but should be based on a master/slave model, the node `0` obviously playing a special part.

Some research projects try to take care of this flaw as in `COBRA` [28] or `PARDIS` [29] where the `IDL` language is extended to handle “distributed” data types.

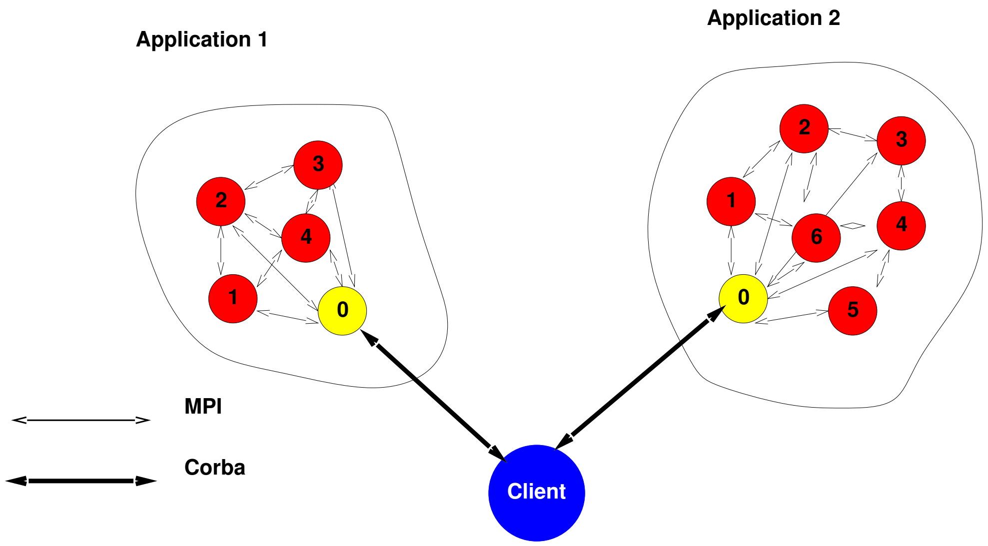


Figure 18: A `Corba` client monitoring `Mpi`-based applications

Anyway a good practice in distributed and parallel programming is to try to reduce communications by improving the “locality” of the algorithms. Clearly the overall performance is tightly connected to the characteristics of the network -bandwidth of course but also, in this context, “startup” time- and the availability of an optimized environment specific protocol for `Corba`.

5.4 Conclusions

Once a server is developed, only a few “generic” additional features are necessary to undertake coupling with another application. Implementation of these new functionalities is in most cases straightforward. Servers can be sequential or parallel applications. If legacy software is encapsulated its original performances are not affected.

The implementation of the coupling itself remains external to the original applications. It is totally contained in the client code (or another secondary module also controlled by the client as in figure 16). Inter application parallelism is also handled in the client code.

Availability of implementations of new paradigms: component model, "parallel objects", will allow more ease and flexibility in designing and implementing distributed parallel applications.

6 Numerical results and opening remarks

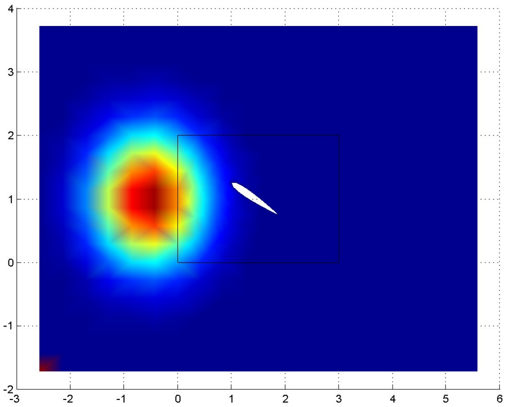
On figure 19, the evolution of the heat blob around the airfoil in the unsteady case (the results are similar to the steady case) with a window of 10 time steps ($\delta t = .05$) is shown. Coupling is done through `Corba`. As can be seen, going through the finite element border does not interfere with the heat blob convection, thus validating in this case the numerical algorithm.
For the first 5 time windows, convergence takes under a 100 iterations, while when the blob leaves the finite element domain, number of iterations is reduced by half. The whole computation takes under 5 minutes, validating the coupling prototype.

To conclude, several remarks can be made:

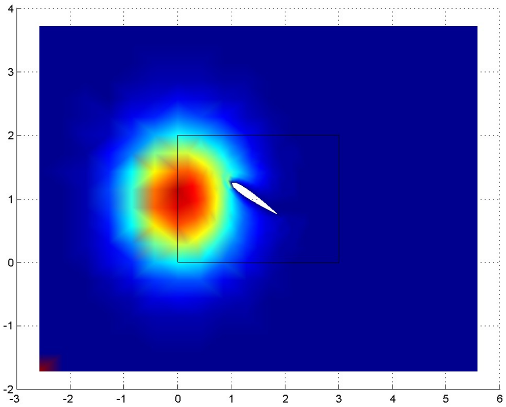
First, one of the key points of the domain decomposition method presented here is its great pliability in terms of border exchanges and local computation, thus allowing better adaptability to the hardware. By adding the time dimension, communications can be adjusted to latency and bandwidth, independently of the space size of the domains. It is particularly well-suited to distributed computing. Connections with high latency and large bandwidth will be most effective with large time windows, and data transfers can be overlapped by local computations. Low latency and small bandwidth systems can work on a the steady coupling (a one time step window) without modifying the precision of the results.

Secondly, this work has shown the feasibility of coupling heterogeneous methods on distant sites. The software technology used is simple and provides immediate access to the codes. In this context the coupling is non-intrusive and is described in an external module.

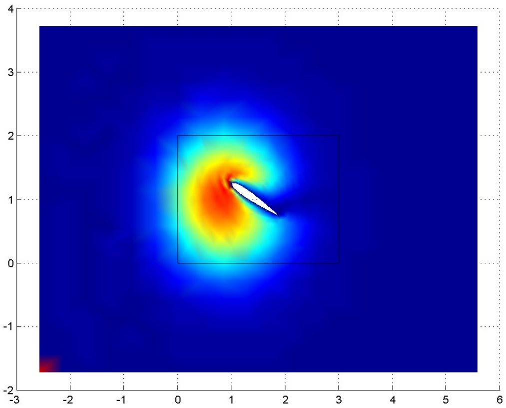
Finally, such techniques allow fast prototyping of new algorithms and methods and is a step towards total flexibility in `CFD` and other fields of numerical computations.



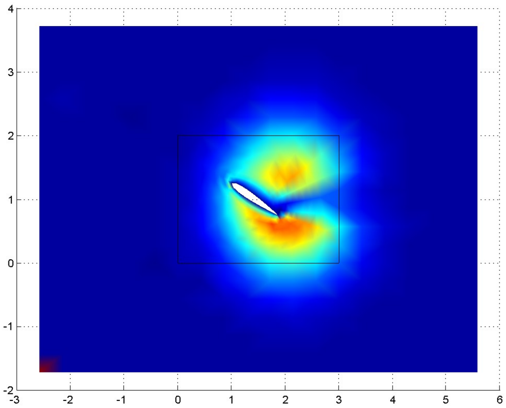
a) T=0



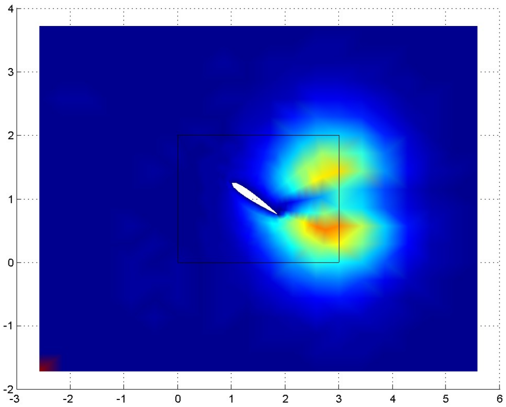
b) T=10 δt



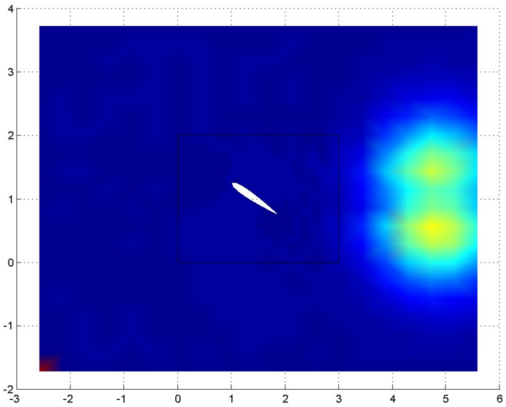
c) T=20δt



d) T=50δt



e) T=70δt



f) T=100δt

Figure 19: Time evolution of a heat blob around a Naca airfoil

References

[1] M. Gander and L. Halpern, *Optimized Schwarz Waveform Relaxation for Convection Reaction Diffusion Problems*, in preparation.

[2] M. J. Gander, L. Halpern, and F. Nataf, *Optimal convergence for overlapping and non-overlapping Schwarz waveform relaxation*, In C-H. Lai, P. Bjørstad, M. Cross, and O. Widlund, editors, Eleventh international Conference of Domain Decomposition Methods, 1999.

[3] L. Halpern & M. Schatzman, *Artificial boundary conditions for incompressible flows*, SIAM Journal on Math. Anal, vol 20, #2, 308-353, 1989.

[4] C. Japhet, *Conditions aux limites artificielles et décomposition de domaine: méthode OO2 (optimisée d'ordre 2). Application à la résolution de problèmes en mécanique des fluides*, Thèse, Ecole Polytechnique, 1997.

[5] C. Japhet, F. Nataf and F. Rogier, *The Optimized Order 2 method. Application to convection-diffusion problems*, Future Generation Computer Systems FUTURE, vol 18, 2001.

[6] F. Nataf, communication personnelle.

[7] F. Nataf and F. Rogier, *Factorization of the Convection-Diffusion Operator and the Schwarz Algorithm*, M^3AS,vol 5 ,#1, pp 67–93,1995.

[8] Grand M., *Java Languare Reference*, 2^{nd} Edition, O'Reilly, 1997, ISBN 1-56592-326-X.

[9] *The Java Programming Language*, `http://java.sun.com/`.

[10] *Java Grande Forum*, information at `http://www.javagrande.org`.

[11] *Java Numerics*, information at `http://math.nist.gov/javanumerics`.

[12] Farley J., *Java Distributed Computing*, O'Reilly, 1998, ISBN 1-565-92206-9.

[13] GRID, *The GRID Forum*, `http://www.gridforum.org/`.

[14] EGRID *The European Grid Forum*, `http://www.egrid.org`.

[15] Chervenak A., Foster I., Kesselman C., Salisbury C., Tuecke S., *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets.*, available on line at [16], to be published in Journal of Network and Computer Applications, 2000.

[16] *The GlOBUS project*, Information at `http://www.globus.org/`.

[17] Chapin S., Karpovich J., Grimshaw A., *The Legion Resource Management System*, Procs. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'99), San Juan, Porto Rico, April 1999.

[18] *The LEGION project at the University of Virginia, USA*, `http://legion.virginia.edu/`.

[19] Siegel J. et al., *CORBA Fundamentals and Programming*, J. Wiley & Sons, 1996, ISBN 0-471-12148-7.

[20] Henning M, Vinoski S., *Advanced CORBA Programming with C++*, Addison-Wesley, 1999, ISBN 0201379279.

[21] *Corba: Common Object Request Broker Architecture*, Information at `http://www.corba.org`.

[22] `OMG` *the Object Management Group*, `http://www.omg.org`.

[23] Stroustrup B., *The C++ programming language*, 3rd Edition, Addison-Wesley, 1998, ISBN 0-201-88954-4.

[24] Barton J., Nackman L., *Scientific and Engineering C++*, Addison-Wesley, 1994.

[25] `OMG CCM` Implementers Group, *CORBA Component Model Tutorial*, Document–ccm/02-04-01, available at `http://www.omg.org`.

[26] Gropp W., Lusk E., Skjellum A., *Using MPI*, 2nd Edition, MIT Press, 1999, ISBN 0-262-57132-3.

[27] *MPI: Message Passing Interface*, All documents can be retrived from `http://www.mpi-forum.org/`. For information and implementations of the standard see `http://www-unix.mcs.anl.gov/mpi/`.

[28] Priol T., René C., Alléon G., *Programming SCI Clusters Using Parallel CORBA Objects*, INRIA-IRISA report 3649, March 1999.

[29] Keahey K., Gannon D., *PARDIS: CORBA-based Architecture for Application-level Parallel Distributed Computation*, Proceedings of Supercomputing'97, November 1997.